# Efficient Algorithms for Crowd-Aided Categorization

Yuanbing Li†, Xian Wu*, Yifei Jin†, Jian Li*, Guoliang Li†
†Department of Computer Science, Tsinghua University, Beijing, China
*Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
yb-li16@mails.tsinghua.edu.cn, {infinity0222, yfjin1990}@gmail.com, {lijian83, liguoliang}@tsinghua.edu.cn

## ABSTRACT

We study the problem of utilizing human intelligence to categorize a large number of objects. In this problem, given a category hierarchy and a set of objects, we can ask humans to check whether an object belongs to a category, and our goal is to find the most cost-effective strategy to locate the appropriate category in the hierarchy for each object, such that the cost (i.e., the number of questions to ask humans) is minimized. There are many important applications of this problem, including image classification and product categorization. We develop an online framework, in which category distribution is gradually learned and thus an effective order of questions are adaptively determined. We prove that even if the true category distribution is known in advance, the problem is computationally intractable. We develop an approximation algorithm, and prove that it achieves an approximation factor of 2. We also show that there is a fully polynomial time approximation scheme for the problem. Furthermore, we propose an online strategy which achieves nearly the same performance guarantee as the offline optimal strategy, even if there is no knowledge about category distribution beforehand. Experiments on a real crowdsourcing platform demonstrate the effectiveness of our method.

**PVLDB Reference Format:**
Yuanbing Li, Xian Wu, Yifei Jin, Jian Li, Guoliang Li. Efficient Algorithms for Crowd-Aided Categorization. *PVLDB*, 13(8): 1221 -1233, 2020.
DOI: https://doi.org/10.14778/3389133.3389139

## 1. INTRODUCTION

During the past few years, crowdsourcing has emerged as a major technique for solving large-scale problems that are considered easy for human beings, but rather difficult for computers to solve solely [33, 18, 14]. Examples include comprehending images or videos, translating natural languages and evaluating search results. Facilitating the interaction between humans and machines in order to better harness the human intelligence has becomes a central research theme. There have been several database systems for incorporating human assistance into data processing tasks [21, 27], including sorting [20], computing maximum [31], counting [19] and finding specific items [5]. Moreover, crowdsourcing has also been used in a variety of applications, such as entity resolution [32, 33, 34, 35], path selection [37], generating planning queries [11], filtering noisy data [12, 24, 26] and scheduling crowdsourcing tasks [8].

In this paper, we focus on the object categorization problem [25, 29], which is another important problem. Given a set of categories and a set of uncategorized (i.e., unlabeled) objects, we would like to inquire the crowd to find the most suitable category (i.e., label) for each object. Each question incurs a certain amount of monetary cost. If the number of categories is large (say $\geq 10,000$), usually a lot of questions are needed to pin down the final category (we cannot possibly list all 10,000 or more options in a single multiple-choice question). Hence, our goal is to find the best question-asking strategy to minimize the overall cost. We assume that the set of categories forms a *hierarchical structure* (for instance, "fish", "birds" are subcategories of "animals"), which is known in advance. This assumption is justified in many real-world applications [25, 29]. Furthermore, the hierarchy naturally defines a set of intuitive (for humans) questions, each corresponding to a node therein.

**Motivating Applications:** Alice is a photographer. She once took a lot of photos during her visit to several cities in Asia. Now she wants to classify the photos into the geographical hierarchy, shown in Figure 1. Due to the large number of photos, she decides to seek help from the crowd. For each photo, she selects a node from the hierarchy, and issues a multiple-choice question as depicted in Figure 2. After collecting the answers, she selects another node and issues a new question based on them. By repeating the procedure she can finally identifies the most suitable category for that photo. Besides the above toy example, our problem applies to, but is not limited to, the following scenarios:

**Product Categorization:** Categories of products naturally form a hierarchical structure. Companies seek to optimize their product taxonomies, which makes their products easier for consumers to find. To achieve this, we can ask the crowd to categorize products according to the hierarchy. More concretely, we can post a picture as well as descriptions of a product, then select a category as the question with all of its subcategories as options. Workers on the crowdsourcing platform should either pick a subcategory for

*Guoliang Li is the corresponding author.

*Proceedings of the VLDB Endowment,* Vol. 13, No. 8
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3389133.3389139

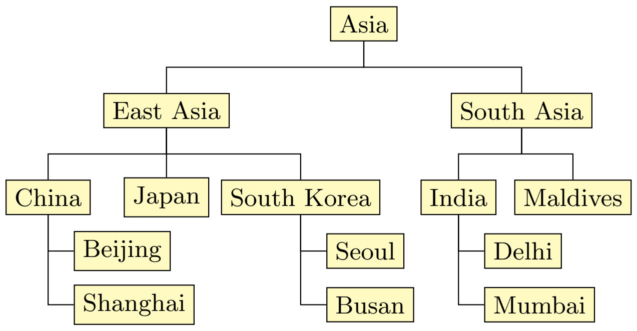
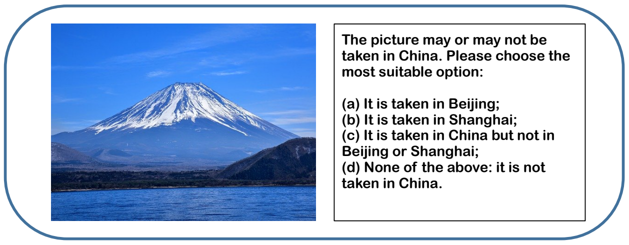


Figure 1: An example of a category hierarchy.



The picture may or may not be taken in China. Please choose the most suitable option:

(a) It is taken in Beijing;
(b) It is taken in Shanghai;
(c) It is taken in China but not in Beijing or Shanghai;
(d) None of the above: it is not taken in China.

Figure 2: An example of a multiple-choice question. This question corresponds to the "China" node in the hierarchy.

this product, or just click the "None of the above" option indicating that it does not belong to this category or any of its subcategories. It is worth noting that Amazon has been posting product categorization tasks on Amazon Mechanical Turk (AMT)[†], which are similar to what we describe above.

**Image Classification:** We would like to categorize the images into the predefined category hierarchy by asking people questions depicted in Figure 2. The task is crucial in supervised learning as it can provide a large amount of labeled data for training and testing. For example, given the hierarchy in Figure 1 and a set of uncategorized photos, the simplest strategy is to travel in the original hierarchy in a top-down manner. Namely, we always ask the first question corresponding to the root of the hierarchy (a question at "Asia"). Depending on the answer, we ask the second question corresponding to a node in the second level of the hierarchy. Although this strategy is fairly reasonable, there may be better ones based on the distribution of categories. Consider the case where most of the photos are taken in South Asia. A better strategy would be to first ask the question at "South Asia" (such strategy incurs an average cost per photo close to 1, while the previous one at least 2).

From the above example, we can see that it is possible to design a strategy better than the one suggested by the original hierarchy, if the category distribution is known beforehand. However, in many situations, it is impossible to know the *a-priori* distribution (for example, Alice cannot remember even roughly how many pictures she took in each country). Moreover, we will see that even if the distribution is known in advance, it is still quite challenging to come up with a plan that minimizes the total cost.

Therefore, based on the observations from the examples above, we propose the following natural but challenging question: *how can we construct a unified plan of asking questions to minimize the overall cost, even if we have no knowledge of the category distribution?*

[†]`http://www.mturk.com/`

Parameswaran *et al.* [25] proposed to utilize human power to categorize objects in trees or directed acyclic graphs. Tao *et al.* [29] introduced *interactive graph search* (IGS) problem, and used heavy path decomposition to reduce the crowd cost. Our work differs from theirs in the following aspects. First, we utilize multiple-choice crowdsourcing questions, which are quite common in practical categorization tasks [6]. Second, we consider the category distribution of the object set, which is more general and can better reflect the actual conditions. Third, we develop strategies for online learning.

**Our Contributions:** The technical contributions of this paper are summarized as follows:
(1) We propose an online framework for categorizing objects in a crowdsourcing platform (Section 2). We model the plan of asking questions as a decision tree construction problem. Then, we prove that the problem is NP-hard. To approximate the optimal decision tree, we develop a simple and intuitive greedy algorithm, and prove that its approximation ratio is 2. Moreover, we develop a non-trivial fully polynomial time approximation scheme (FPTAS) [‡] for this problem (Sections 2 & 3).
(2) We present an online learning algorithm, based on *following-the-perturbed-leader*. The algorithm gradually learns the true distribution of the object set and can achieve a nearly optimal performance in the worst case, compared to the best offline optimal decision tree in hindsight (Section 4).
(3) We propose two extensions: (*i*) we study how to batch and issue multiple questions to reduce the latency; (*ii*) To better handle the situation where the objects arrive in a stream fashion and the category distribution changes over time, we develop methods for adaptively tracking the current category distribution (Section 5).
(4) We conduct extensive evaluations on the proposed strategies and compare the performance with the state-of-the-art method, on both synthetic and real data. Our results demonstrate that our strategies can lead to more than 40% cost savings on a real crowdsourcing platform (Section 6).

## 2. CROWD-AIDED CATEGORIZATION

### 2.1 Problem Formulation

Let $\mathcal{O} = \{o_1, \dots, o_n\}$ be a set of $n$ objects. Let $\mathcal{T}$ denote a given category hierarchy, which can be represented as a tree, in which each node corresponds to a distinct category. For ease of presentation, for a node $u \in \mathcal{T}$, we also use $u$ to represent the corresponding category. For any object $o \in \mathcal{O}$, we denote $\mathsf{tar}(o)$ as its target (the most suitable) category, i.e., $\mathsf{tar}(o)$ has no suitable descendant category to which object $o$ belongs. For instance, the target category of the photo in Figure 2 is "Japan". For each node $u \in \mathcal{T}$, we use $\mathcal{T}_u$ to denote the subtree rooted at $u$, $\mathsf{child}_{\mathcal{T}}(u)$ to denote the set of children of $u$, $\mathsf{father}_{\mathcal{T}}(u)$ to denote the father node of $u$ and $\mathsf{root}(\mathcal{T})$ to denote the root of $\mathcal{T}$.

**Crowd Question.** Next we formally define the form of the questions to be asked on crowdsourcing platforms:

[‡]An algorithm $A$ is an FPTAS for an optimization problem $P$, if for any instance $\mathcal{I}$ and any constant $\epsilon > 0$, $A$ computes a solution $S$ in polynomial time *w.r.t.* the problem size $n$ and $1/\epsilon$, of which the value of $S$ satisfies $|\mathsf{OPT} - \mathsf{val}(A)| \le \epsilon\mathsf{OPT}$, where $\mathsf{OPT}$ stands for the value of the optimal solution and $\mathsf{val}(A)$ is the solution returned by $A$.

Definition 1 (Multiple-choice Question). *Given an object $o \in \mathcal{O}$ and an* internal *node $u \in \mathcal{T}$, we request the crowd to answer a multiple-choice question on node $u$, by selecting an option from the following ones:*

- Each $v \in \mathsf{child}_{\mathcal{T}}(u)$ has a corresponding option: The object belongs to category $v$ (i.e., $\mathsf{tar}(o) \in \mathcal{T}_v$);
- The object does not belong to any subcategory of $u$, but does belong to category $u$ itself (i.e., $\mathsf{tar}(o) = u$);
- ("None of the above" option) The object does not belong to $\mathcal{T}_u$ *(i.e.,* $\mathsf{tar}(o) \notin \mathcal{T}_u$*).*

If the option of the second type is selected, the target category is determined and no more questions are needed. While for options of the first or third type, we may have to proceed with further questions to locate the target category.

Example 1. *Given the hierarchy depicted in Figure 1, the question corresponding to node "China" is shown at the right half of Figure 2 with 4 options, of which the first two options are of the first type (subcatgories), and the last two are of the second (itself) and third type (none of the above) respectively.*

A photo of *Great Wall* should lead to the option (a) Beijing, i.e., its target category is "Beijing", while a photo of *Oriental Pearl Tower* should lead the option (b) Shanghai. If Alice posts a photo of *Terracotta Warriors and Horses*, she would receive the answer (c), which means that the target category for this photo is "China". In fact, the photo was taken in Xi'an, which is a northwestern city in China but not included in the given hierarchy. Thus "China" should be the most suitable category. Finally, if the photo is about *Mount Fuji* as shown at the left half of Figure 2, which is in Japan, the correct option would be (d) "None of the above". Then a couple of more questions are needed to determine its target category.

Assume that each question costs a fixed price (for instance, \$0.02 per question), the total *monetary* cost is proportional to the total number of questions needed for categorizing all objects in $\mathcal{O}$. Hence, our goal is to minimize the total number of questions.

Problem 1 (Cost Minimization). *Find an online adaptive question-asking strategy to categorize all objects in $\mathcal{O}$ such that the total number of questions is minimized.*

**Remark:** Some nodes in the hierarchy may have a large number of children (for instance, the "Books" category in Amazon has more than 30 subcategories). Therefore, it may be unfriendly for workers to choose among such a large number of options. In practice, options can be displayed in groups (for example, every 10 options make a group). When a worker eliminates all options of a group, another group comes successively. In this way the practicality of our framework is guaranteed. Another possible solution is to use the binary version of our framework, where all questions on crowd are *yes-or-no* questions rather than multiple-choice questions and thus there are no abundant subcategories of such nodes. We defer the detail of modification to the evaluation part (Section 6).

## 2.2 Framework and Workflow

We face two challenges: (a) the complex structure of the hierarchy makes it hard to develop a cost-saving strategy, even if the *a-priori* probability distribution is already in hand; (b) In many scenarios, the *a-priori* probability distribution is unknown in advance. To address these challenges, we first present the design of our algorithmic framework, which is depicted in Figure 3. Our framework takes as input a set of objects, which can be very large, and a category hierarchy without distribution information, consisting of three key components: (1) the learning component for learning the probability distribution, (2) the optimization component for constructing a question-asking plan given the estimated distribution, and (3) the crowd responsible for interacting with crowdsourcing platforms on generating tasks and answers.

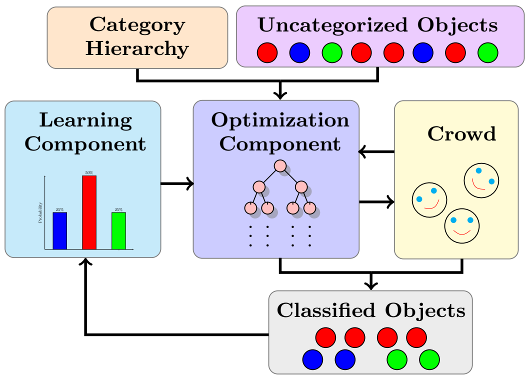


Figure 3: An online framework.

**Algorithm 1** Framework($\mathcal{O}, T$)

```
Input: $\mathcal{O}$: objects; $\mathcal{T}$: category hierarchy
Output: $\mathcal{L}=\{(o_i, \mathsf{tar}(o_i))|1 \le i \le n\}$: categorized objectsaggregating
 1: $\mathcal{L} \leftarrow \emptyset$
 2: Set the initial category distribution
 3: Generate a plan $\mathcal{P}$ of asking questions
 4: while there is an uncategorized object $o$ in $\mathcal{O}$ do
 5:     while $\mathsf{tar}(o)$ is not determined do
 6:         Ask a question to the crowd following $\mathcal{P}$
 7:         Collect answers from the crowd
 8:     end while
 9:     $\mathcal{L} \leftarrow \mathcal{L} \cup \{(o, \mathsf{tar}(o))\}$
10:     Re-estimate the category distribution
11:     Reconstruct the plan $\mathcal{P}$
12: end while
13: return $\mathcal{L}$
```

Algorithm 1 shows the pseudo code of the online framework. Each time the framework takes in an uncategorized object from $\mathcal{O}$ (Line 4). It follows the question-asking plan constructed by the optimization component, proceeds with iteratively generated new tasks and collecting answers from the crowd, until the target category for this object is determined (Lines 5-9). The learning component then reestimates the distribution based on the target categories of the finished objects (Line 10). Finally the optimization component adjusts the plan of asking questions according to the re-estimated distribution to minimize the cost (Line 11).

Note our framework works in the online setting, in the sense that the question-asking plan for the current object should only depend on the objects we have processed so far, regardless of those unprocessed ones. We do not even assume that all objects are given before our framework starts. They may arrive in an online stream fashion. For each object, our question-asking plan can be *adaptive*. Namely, the next question to ask may depend on the results of the previous ones. In next section, we show this is in fact equivalent to and implemented by constructing a new decision tree based on the original given hierarchy and results of categorized objects so far.

## 2.3 Adaptive Strategies and Decision Trees

As stated in Section 2.1, an internal node $u \in \mathcal{T}$ corresponds a multiple-choice question, which may have three types of options. By resorting to the crowd, we get the answer, and select another node to ask based on the answer. In fact such interaction with the crowd corresponds the partitioning and pruning operation on the given hierarchy $\mathcal{T}$. Let us elaborate the process in graph theoretical terms and use the hierarchy depicted in Figure 1 as an example. Suppose the internal node $u \in \mathcal{T}$ ("China") is selected. If we remove all the edges incident on it, the hierarchy $\mathcal{T}$ is partitioned by $u$ to the following subtrees:

1. One subtree for each child of $u$, which corresponds to the options of the first type in Definition 1: the subtree "Beijing" and "Shanghai" of size 1.
2. Node $u$ itself, which corresponds to the second type option: the subtree "China" which contains only itself of size 1.
3. One remaining subtree containing the parent of $u$, which corresponds to the "None of the above" option: the subtree rooted at "Asia" but in which "East Asia" has only "Japan" and "South Korea" as its children.

We use $\Phi_{\mathcal{T}}(u)$ to denote the set of subtrees obtained above. Hence, the answer from the crowd to the question can help us to identify which subtree in $\Phi_{\mathcal{T}}(u)$ contains the target category. Based on the answer, it is natural to explore the corresponding subtree and proceed to next node/question in it if needed, and all other subtrees are thus eliminated. By repeating the procedure, the size of search space is tightened iteratively and the target category is determined when there is only one node left.

Such an adaptive strategy (i.e., implemented in the optimization component) can be modeled as a *decision tree*, in which each internal node is a (multiple-choice) question (say it corresponds to $u \in \mathcal{T}$), whose ancestors are the previously asked questions, and each branch of it directing to one subtree in $\Phi_{\mathcal{T}}(u)$ connects to a question therein as a child node. And all leaves of the decision tree are potential target categories. For a leaf node $l$, the root-to-leaf path in the decision tree represents the sequence of questions leading to the target category, and the cost spent for the object $o$ with the leaf node as its target category ($\mathsf{tar}(o) = l$) is the depth of the leaf node (i.e., the length of the path minus 1) in the decision tree. Please refer to Figure 4 for an example.

Now, we are ready to formally define the decision tree constructed by the adaptive strategy.

Definition 2 (Decision Tree). *Given the initial hierarchy $\mathcal{T}$, a* decision tree $\mathcal{D}$ w.r.t. $\mathcal{T}$ *satisfies:*

- *Each leaf of $\mathcal{D}$ is a category in $\mathcal{T}$;*
- *Each internal node $v$ of $\mathcal{D}$ corresponds to an internal node $u \in \mathcal{T}$. $u$ partitions $\mathcal{T}$ into $\Phi_{\mathcal{T}}(u)$, and thus $v$ has $|\Phi_{\mathcal{T}}(u)|$ children. Each child $w \in \Phi_{\mathcal{T}}(u)$ corresponds to a subtree in $\Phi_{\mathcal{T}}(u)$, and $\mathcal{D}_w$ (i.e., the subtree of $\mathcal{D}$ rooted at $w$) represents the decision tree for the corresponding subtree.*

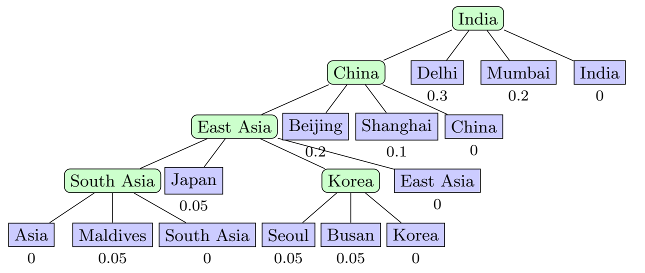


Figure 4: Optimal decision tree to the category tree in Figure 1. Question nodes are represented by green round rectangles and leaves (potential target categories) are represented by purple rectangles. Category probabilities are labeled below the corresponding leaves.

Based on the definition, it is trivial to see that given any tree $\mathcal{T}$, the size of its decision tree $\mathcal{D}$ is at most twice the size of $\mathcal{T}$, since every node of $\mathcal{T}$ appears as a leaf node (target category) of $\mathcal{D}$ whereas at most all the internal nodes of $\mathcal{T}$ serve as internal nodes (multiple-choice questions) of $\mathcal{D}$.

For ease of presentation, for each internal node $v \in \mathcal{D}$, we fix its *leftmost* subtree as the decision tree for the subtree corresponding to the "None of the above" option in Definition 1, and its *rightmost* child as the target category corresponding to the question itself (i.e., the second type option). For instance, in Figure 4, the question node at "India" has 4 children, with the question node "China" as its leftmost child and the target category "India" as its rightmost child.

Let $\mathsf{Lvs}(\mathcal{D})$ denote the set of leaves of $\mathcal{D}$. For ease of presentation, for any $u \in \mathcal{T}$, we also use $u$ to denote the question (internal) node in $\mathcal{D}$ asked at $u$[§]. For any $u \in \mathcal{D}$, denote by $\mathsf{dep}_{\mathcal{D}}(u)$ the depth of $u$ in $\mathcal{D}$.

In order to define the cost of a decision tree, we first define the category distribution as follows: Let Pr be the probability distribution over category set $\mathcal{T}$, defined by

$$\Pr(u) \triangleq |\{\mathsf{tar}(o) = u \mid o \in \mathcal{O}\}|/|\mathcal{O}|.$$

We call Pr the category distribution. For $\mathcal{T}' \subseteq \mathcal{T}$, we define $\Pr(\mathcal{T}') \triangleq \sum_{u \in \mathcal{T}'} \Pr(u)$. Now, we can define our cost metric.

Definition 3 (Cost Metric). *Given a decision tree $\mathcal{D}$, the expected cost, denoted by $\mathsf{cost}(\mathcal{D})$, is*

$$\mathsf{cost}(\mathcal{D}) \triangleq \frac{1}{|\mathcal{O}|} \sum_{o \in \mathcal{O}} \mathsf{dep}_{\mathcal{D}}(\mathsf{tar}(o)) = \sum_{u \in \mathsf{Lvs}(\mathcal{D})} \Pr(u)\, \mathsf{dep}_{\mathcal{D}}(u) \tag{1}$$

Problem 2 (Min-cost Decision Tree (MDT)). *Given $(\mathcal{T}, \Pr)$, construct a decision tree $\mathcal{D}^*$ w.r.t. $\mathcal{T}$ with the minimum expected cost.*

**Remark:** By the definition of a decision tree, we can treat the initial category hierarchy $\mathcal{T}$ as a slightly "incomplete" decision tree. The only difference is that any internal node in a decision tree has a leaf node as its child corresponding to the second type option (in Definition 1). Therefore, we can easily extend $\mathcal{T}$ to a decision tree by adding such leaf nodes. In this decision tree, we always start from the root of the hierarchy. Therefore, the "None of the Above" option of any question is unnecessary since it would always lead to

[§]In the paper, for any category $u$, we use $u$ as its corresponding node in $\mathcal{T}$, and a question node asked at $u$ in $\mathcal{D}$.

an empty set. For instance, consider a photo of the Great Wall and the hierarchy in Figure 1. Alice can consecutively ask questions at "Asia", "East Asia", "China" and finally obtain "Beijing" as its target category.

Example 2. *For an object set with the category distribution shown in Figure 4, the decision tree directly obtained from Figure 1 incurs an expected cost of 2.9, while the expected cost of the optimal decision tree $\mathcal{D}^*$ for this hierarchy is 1.85, as depicted in Figure 4.*

## 3. DECISION TREE CONSTRUCTION

In this section, we focus on the MDT problem. We assume the category probability Pr is already known in this section. We first prove that constructing the optimal decision tree is computationally intractable. Then we propose a natural greedy approach and analyze its approximation ratio. Finally, we propose a non-trivial FPTAS for the problem.

### 3.1 NP-Hardness

To prove that constructing an optimal decision tree is NP-hard, we utilize a non-trivial polynomial time reduction from the Exact Cover by 3-Sets with multiplicity 3 problem (X3C-3), defined as follows:

Definition 4 (X3C-3). *An instance of the X3C-3 problem $\mathcal{I} = (X, \mathcal{Y})$ contains the following:*

*(a) a finite set $X$ with $|X| = 3q$ for $q \in \mathbb{N}^+$;*
*(b) a collection $\mathcal{Y}$ of 3-element subsets of $X$, i.e., $\mathcal{Y} = \{Y_1, \ldots, Y_{|\mathcal{Y}|}\}$, and for each $i = 1, \ldots, |\mathcal{Y}|$, it holds that $Y_i \subseteq X$ and $|Y_i| = 3$.*
*(c) each element $x \in X$ appears in at most 3 sets of $\mathcal{Y}$.*

Definition 5 (X3C-3 Problem). *Given an instance $\mathcal{I} = (X, \mathcal{Y})$, the X3C-3 problem decides whether $\mathcal{Y}$ contains an exact cover for $X$, i.e., a sub-collection $\mathcal{Y}' \subseteq \mathcal{Y}$ such that members of $\mathcal{Y}'$ form a disjoint partition of $X$.*

The X3C-3 problem is known to be NP-complete [4]. We now state the following theorem. Please refer to our technical report [1] for the complete proof.

Theorem 1. *The decision version of MDT problem is NP-complete.*

Proof (Sketch). For any instance $\mathcal{I} = (X, \mathcal{Y})$ to the X3C-3 problem, we can construct a category hierarchy $\mathcal{T}$ correspondingly, which contains $|\mathcal{Y}|$ *components*, (see definition in the complete proof). Each component contains nodes associated with elements in $Y_i$ and 5 additional nodes. Denote by $\mathcal{S}^*$ and $\mathcal{D}^*$ the optimal solution to $\mathcal{I}$ and the optimal decision tree to $\mathcal{T}$, respectively. By carefully assigning probabilities to the nodes, we prove that for each component, there exist only two possible decision components in $\mathcal{D}^*$, either including elements of $Y_i$ or excluding them. Moreover, each set $Y' \in \mathcal{S}^*$ corresponds to a component *excluding* elements of $Y'$ in $\mathcal{D}^*$ while each set $Y_0 \in \mathcal{Y} - \mathcal{S}^*$ corresponds to a part *including* elements of $Y_0$ in $\mathcal{D}^*$. By establishing a close relation between these two problems we can prove that MDT is NP-complete. □

**Algorithm 2** Greedy($\mathcal{T}$, Pr)

```
Input: $\mathcal{T}$, Pr
Output: $\mathcal{D}$
 1: $\mathcal{D} \leftarrow \emptyset$
 2: Queue $S \leftarrow (\mathcal{T}, \emptyset)$
 3: while $|S| > 0$ do
 4:     $(\mathcal{T}_0, u_0) \leftarrow S.dequeue()$
 5:     $v \leftarrow \emptyset$
 6:     $min \leftarrow +\infty$
 7:     for $u \in \mathsf{Int}(\mathcal{T}_0)$ do
 8:         $t \leftarrow$ heaviest weight of $\Phi_{\mathcal{T}_0}(u)$
 9:         if $t < min$ then
10:             $min \leftarrow t$
11:             $v \leftarrow u$
12:         end if
13:     end for
14:     if $\mathcal{D} = \emptyset$ then
15:         Assign $\mathsf{query}(v)$ as the root of $D$
16:     else
17:         Assign $\mathsf{query}(v)$ as a child of $u_0 \in D$
18:     end if
19:     for $u \in \mathsf{child}_{\mathcal{T}}(v)$ do
20:         if $u$ is a leaf then
21:             Assign category $u$ as a leaf child of $\mathsf{query}(v)$
22:         else
23:             $S.enqueue(\mathcal{T}_u, \mathsf{query}(v))$
24:         end if
25:     end for
26:     Assign category $v$ itself as a leaf child of $\mathsf{query}(v)$
27:     $S.enqueue(\mathcal{T}_0 - \mathcal{T}_v, \mathsf{query}(v))$
28: end while
29: return $D$
```

### 3.2 Greedy Strategy

To minimize the expected cost, a "good" strategy should first issue questions at nodes that are capable of partitioning the category hierarchy fairly evenly (w.r.t. the distribution Pr). Intuitively, this could avoid the case where a node with a high probability is located deeply in the decision tree (which would incur a large cost). For example, if a node $u$ has probability $\Pr(u) > 0.9$, it is wise to first ask the question that can directly lead to $u$ (i.e., the question at node $u$ if $u$ is an internal node of $\mathcal{T}$, or $\mathsf{father}_{\mathcal{T}}(u)$ if $u$ is a leaf). A heuristic approach is to choose a question for some category $u_0$ which makes the heaviest resulting subtree among $\Phi_{\mathcal{T}}(u_0)$ as light as possible (regard the total probability of a subtree as its weight). Formally, given an instance $(\mathcal{T}, \Pr)$ to the problem, denoting by $\mathsf{Int}(\mathcal{T})$ the set of internal nodes of $\mathcal{T}$, we choose an internal node $u_0$ such that

$$u_0 = \arg \min_{u \in \mathsf{Int}(\mathcal{T})} \max\{\Pr(\mathcal{T}') | \mathcal{T}' \in \Phi_{\mathcal{T}}(u)\} \qquad (2)$$

For ease of analysis, we assume that distinct subtrees of $\mathcal{T}$ have different probabilities, to guarantee the uniqueness of each choice.

**Remarks:** (1) This natural greedy algorithm is much similar to the classic Huffman Algorithm, not only in the problem formulation, but also in the sense that they both try to *balance* the decision tree. The main difference is that our approach is to construct a tree from top to bottom while the Huffman tree is constructed in a bottom-up manner. (2) Prior to our work, Parameswaran *et al.* [25] proposed a graph-based categorization problem, in which they asked *yes-or-no* questions and developed algorithms to rule out as many candidates as possible in each phase *locally* in order to save money. However, in this paper, we focus on minimizing the overall cost, which can be seen as a *global* objective. Moreover, they assumed that the category distribution was *uniform* while we do not require any knowledge of the category distribution.

Please refer to Algorithm 2 for the pseudo-code of the greedy algorithm. It constructs the greedy decision tree from top to bottom in a breadth-first manner. We first initialize an operating queue $S$ in Line 2. The main loop does not terminate as long as there are subtrees to be partitioned (Lines 3-28). Each time we obtain a subtree from the queue (Line 4), traverse all the internal nodes in it, and select the question asked at $v$ ($\mathsf{query}(v)$) according to Equation (2) (Lines 5-13), which partitions $\mathcal{T}_0$ into $|\Phi_{\mathcal{T}_0}(v)|$ parts. Then we place $\mathsf{query}(v)$ to the appropriate position (Lines 14-18), For each subtree of $\mathcal{T}_v$ (recall the options of the first type in Definition 1), it is assigned as a leaf node if it is a single node; otherwise, it is pushed into the queue $S$ for further operations (Lines 19-25). We also assign the category $v$ itself (the second type) as a leaf child of $\mathsf{query}(v)$, and push $\mathcal{T}_0 - \mathcal{T}_v$ (the third type) to the queue $S$ as the leftmost subtree (Lines 26-27).

**Time Complexity:** Denote by $n$ the size of $\mathcal{T}$. In every main loop, the weight of each possible resulting subtree in the sub-hierarchy can be pre-computed in $O(n)$. For any node $u$, it takes $O(|\mathsf{child}_{\mathcal{T}}(u)|)$ time to calculate the weight of the heaviest resulting subtree. Therefore, we can traverse the sub-hierarchy once to obtain the question node. Since there are at most $|\mathsf{Int}(\mathcal{T})|$ sub-hierarchies, the running time of Greedy is $O(n^2)$.

**Space Complexity:** $O(n)$.

The approximation ratio of the greedy algorithm is guaranteed by the following theorem, of which the proof is deferred to our technical report [1].

THEOREM 2. *Given any instance $(\mathcal{T}, \Pr)$ to the MDT problem, let $\mathcal{D}$ denote the decision tree constructed by* Greedy. *It holds that*

$$cost(\mathcal{D}) \leq 2cost(\mathcal{D}^*) \qquad (3)$$

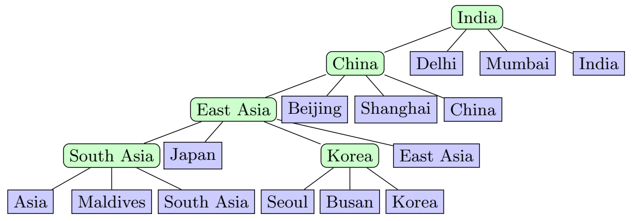
*where $\mathcal{D}^*$ is the optimal decision tree to the problem.*

## 3.3 FPTAS

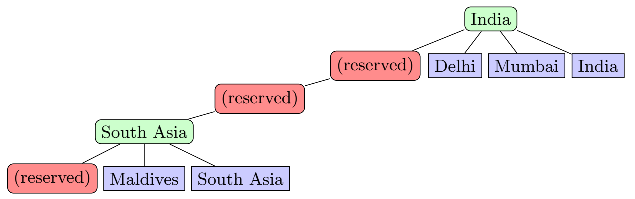
In this section, we present an FPTAS for the problem, which can achieve $(1+\epsilon)$ approximation ratio for any $\epsilon > 0$. For this purpose, we first develop an exact pseudo-polynomial time algorithm based on dynamic programming (DP). Then we show that by rounding the probability of each node in the category tree, the DP can be transformed into an FPTAS.

**Intuitions:** A natural idea to tackle the problem with DP is to first construct the optimal decision tree (the one with the smallest cost) for all possible subtrees of $\mathcal{T}$ and then somehow merge them into the optimal tree for the entire problem. For instance, to generate the optimal decision tree for "Asia" in Figure 1, it would be very tempting to first construct the optimal decision trees for "South Asia" and "East Asia", (see Figure 5b, 5c respectively) and then merge them by taking up the corresponding *reserved* positions (e.g., "India" in Figure 5b corresponds to the reserved root node in Figure 5c) from the other part.

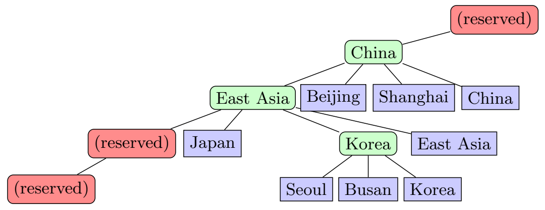
Here we only state the key results. Please refer to the technical report [1] for details of this section.



(a) Optimal decision tree for "Asia"



(b) Decision part for "South Asia"



(c) Decision part for "East Asia".

Figure 5: Subproblems of dynamic programming

THEOREM 3. *Given any instance $(\mathcal{T}, \Pr)$, there exists a pseudo-polynomial time dynamic programming-based algorithm that constructs the optimal decision tree.*

THEOREM 4. *Given any instance $(\mathcal{T}, \Pr)$, there exists an FPTAS for the MDT problem.*

## 4. LEARNING A-PRIORI PROBABILITIES

In this section, we show the mechanism of the learning component in our framework as depicted in Figure 3. Our goal is to develop an online learning algorithm which can gradually learn the category distribution as processing the objects so that the overall long-term online cost is close to the offline optimal cost. The learning component and optimization component interact with each other in a beneficial way. Each time there comes newly categorized objects from the optimization component, the learning component re-estimates the category distribution based on them, which will be used by the optimization component to construct the decision tree next time. Thus, we can iteratively improve the decision tree, and enhance its *adaptivity* for future objects, without knowing their actual distribution in advance.

One natural idea for this problem is to use the *empirical category distribution*. More precisely, we estimate the probability of a category $u \in \mathcal{T}$ as the fraction of the objects whose target category is $u$ among all the processed ones. While this approach is intuitive ¶, it may lead to a rather expensive cost in the worst case with the presence of an *adversarial* input sequence of objects, which is illustrated in the following example:

¶If we can randomly permute all the objects, we can show that the true distribution can indeed be learned according to the law of large numbers. However, the objects may come in an online fashion and a random permutation pre-processing step may not be always possible.

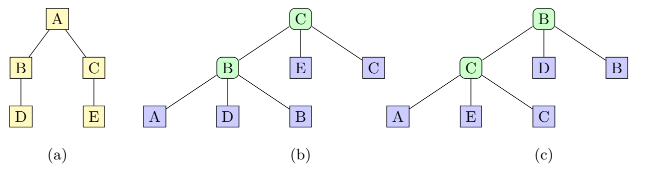


Figure 6: (a) the input category hierarchy; (b) the decision tree in odd times; (c) the decision tree in even times;

EXAMPLE 3. *Consider the input hierarchy shown in Figure 6a. The objects with $D$ or $E$ as target category compose the object set, where the number of objects with target category $D$ is slightly smaller than that of the others. The initial estimated category distribution is set as follows:* $(\Pr(A), \Pr(B), \Pr(C), \Pr(D), \Pr(E)) = (0, 0, 0, \alpha, 1-\alpha)$ *for some* $0 < \alpha < \frac{1}{2}$.

Each time, we take in one object from the set and iteratively ask a question on the crowdsourcing platform until we find the target category for the object. When the objects come in the sequence of $D, E, D, E, \ldots$, it can be seen that we would be totally mislead by the leading object every single time. That is, the decision tree constructed for odd times would be the one shown in Figure 6b (category $E$ is placed at a smaller depth) while that for even times in Figure 6c. After taking in $N$ objects for some sufficiently large $N$, the total cost of this strategy becomes $2N$. However, if we know the *a-priori* probability in advance, the offline optimal decision tree is the one shown in Figure 6b, yielding the cost of $1.5N \ll 2N$.

We modify this strategy by *following the perturbed leader* (FPL): we maintain the fraction of each node in the hierarchy, and each time we estimate the probability as the fraction plus an exponentially distributed perturbation [10].

The workflow of the algorithm is depicted in Algorithm 3. In each loop, we take in an uncategorized object, calculate the weight of each node as the number of categorized objects on that node plus a random variable $\sim \mathsf{exp}(\lambda)$ (Lines 2-5) and normalize the weights as probabilities in Line 6. Then we construct the decision tree using the greedy algorithm or the FPTAS in Section 3 (Line 7). We iteratively ask a question and collect its answer following the decision tree, until the target category is determined (Lines 8-9). Finally, we update the corresponding counter in Line 10.

**Algorithm 3** PerturbedLeader($T, \mathcal{O}, \lambda$)

```
Input: $T, \mathcal{O}, \lambda$
 1: while there is an uncategorized object in $o \in \mathcal{O}$ do
 2:     for $u \in T$ do
 3:         Choose $q(u) \sim \mathsf{exp}(\lambda)$
 4:         $w(u) \leftarrow c(u) + q(u)$
 5:     end for
 6:     Normalize the weight $w$ of each node as probability
 7:     Construct the decision tree $\mathcal{D}$ by Greedy (or FPTAS)
 8:     Ask questions on the crowd based on $\mathcal{D}$
 9:     Collect the target category $\mathsf{tar}(o)$
10:     $c(\mathsf{tar}(o)) \leftarrow c(\mathsf{tar}(o)) + 1$
11: end while
```

In fact, Algorithm 3 realizes the online framework in Section 2. The time cost for categorizing the object set consists of two parts: (a) the total latency incurred on the crowdsourcing platform; (b) the running time of the decision tree construction algorithm and FPL, $O(n^2)$ per object for Greedy and $poly(n/\epsilon)$ for the FPTAS given $\epsilon > 0$.

Recall that the goal of the framework is to adaptively develop efficient algorithms, such that the overall online cost is close to the overall optimal cost given the *a-priori* distribution, even in the *adversarial* setting (e.g., the sequence in Example 3). Fortunately, when FPL as the learning component is combined with the FPTAS as the optimization component, the expected online cost (on the distribution of the perturbation) is quite close to the offline optimal total cost, as shown in the following theorem:

THEOREM 5. *Given any object set $\mathcal{O}$ and any constant $\lambda > 0$, PerturbedLeader (using FPTAS in line 7) achieves:*

$$\mathbb{E}\{cost\} \leq (1+\lambda)\mathsf{offline}^*(\mathcal{O}) + O\left(\frac{n^2 \ln n}{\lambda}\right) \quad (4)$$

*where* $\mathbb{E}\{\mathsf{cost}\}$ *is defined as the expectation of our online cost and* $\mathsf{offline}^*(\mathcal{O})$ *is the offline optimal total cost.*

Please refer to the technical report [1] for the proof.

## 5. EXTENSIONS

In this section, we discuss some useful extensions to our basic framework. First, we propose a batched approach to further reduce the latency. More concretely, instead of asking one question at a time for the object being processed and finishing the input objects one by one, we can batch multiple questions and issue them together. After obtaining the answers, we can proceed to further questions based on them. Second, we consider the situation where the category distribution may change over time.

### 5.1 Batching Multiple Questions

In this section, we explore two dimensions of batching: One is that we may process several objects simultaneously with a shared decision tree, which is called a *stage*. Let $m$ denote the number of objects in each stage. The other is to ask multiple questions about the same object in every *round* of interacting with the crowd. Let $k$ denote the number of questions asked at one object in each round.

For example, consider the hierarchy in Figure 1. For the first dimension of batching, three questions could be issued together for three different photos respectively, one about the Great Wall, one about Mount Fuji, and the last about the Taj Mahal. For the second dimension, we could ask questions at "India", "China" and "Japan" for the same photo concurrently.

**Workflow:** Algorithm 4 shows the workflow. During each stage the framework takes in at most $m$ uncategorized objects in Line 2, estimates the category distribution using FPL in Line 3, and adaptively constructs the shared decision tree using Greedy or the FPTAS in Line 4. Afterwards, for each object taken in, it generates at most $k$ questions in each round until the target category is obtained (Lines 5-9). Finally, it records the results in Line 10.

It is worth noting that an adaptive strategy for asking multiple questions in each round can once again be modeled as a decision tree, in which each internal node represents a set of $k$ questions (each corresponding to a different internal node in $\mathcal{T}$). By Theorem 1, given $k \geq 1$, to construct the optimal decision tree in general is NP-hard. In the remainder of the section, we propose a heuristic algorithm.

**Algorithm 4** BatchedFramework($\mathcal{T}, \mathcal{O}, \lambda, m, k$)

```
Input: $\mathcal{T}, \mathcal{O}, \lambda, m, k$
1: while there are uncategorized objects in $\mathcal{O}$ do
2:     Take in at most $m$ objects from $\mathcal{O}$
3:     Estimate the distribution by FPL($\lambda$)
4:     Construct the decision tree $\mathcal{D}$ by Greedy (or FPTAS)
5:     while these objects are not finished do
6:         Generate $k$ questions for each object based on $\mathcal{D}$
7:         Post the questions on the crowdsourcing platform
8:         Collect answers
9:     end while
10:    Record the results of the categorized objects
11: end while
```

**A Greedy Strategy** We propose a greedy algorithm similar to the one in the previous section: we try to select $k$ internal nodes that make the heaviest resulting subtree as light as possible. Let $U_0 = \{u_1^0, \ldots, u_k^0\}$ denote the set of selected nodes. The strategy is described as follows:

$$U_0 = \arg \min_{U \subseteq \mathsf{Int}(\mathcal{T}), |U|=k} \max\{\Pr(\mathcal{T}') | \mathcal{T}' \in \Phi_{\mathcal{T}}(U)\} \tag{5}$$

where $\Phi_{\mathcal{T}}(U)$ is the set of subtrees obtained by partitioning $\mathcal{T}$ and removing all the edges incident on any node in $U$.

However, it is not trivial to select the $k$ question nodes in each round. A naïve strategy would traverse the whole search space, incurring at least $O(\binom{n}{k})$ running time.

To solve the problem, we can first solve a closely related problem: to select the minimum number of questions (internal) nodes such that the weight of the heaviest resulting subtree does not exceed a given constant $w \in (0, 1]$. Reasonably, a larger number of question nodes may result in a smaller weight of the heaviest resulting subtree. Using the algorithm for the above problem as a subroutine, we can solve the original problem by simply performing a binary search (identifying the smallest $w$ such that the number of required questions is at most $k$) [‖].

Formally, the related problem is defined as follows:

Problem 3 (Partition). *Given ($\mathcal{T}$, Pr) and some constant $w \in (0, 1]$, select the minimum number of question (internal) nodes such that for each resulting subtree, its total probability does not exceed $w$.*

Algorithm 5 shows the pseudo code for the partition problem, and the input is the triple ($\mathcal{T}$, Pr, $w$). It utilizes a greedy strategy: it traverses the tree from the leaf level (the *level* of a node equals its depth plus one.) to the root (Lines 2-10), and the technique of pre-computing weights is utilized again. Each time it meets a node $u$, it calculates the total probability of the subtree $\mathcal{T}_u$ and checks if it exceeds the threshold $w$ (Line 4). If so, it selects $u$ as a question, removes $u$ from the hierarchy $\mathcal{T}$ and adjust the corresponding weights (Lines 5-7).

**Time Complexity:** For each node $u$, we only need to calculate the total probability of $\mathcal{T}_u$, which takes $O(|\mathsf{child}_{\mathcal{T}}(u)|)$. Therefore, the time complexity is $O(n)$.

**Space Complexity:** $O(n)$.

The correctness of the algorithm is guaranteed by the following theorem, and the proof is in the technical report [1].

[‖]We are inspired by the ideas in [25] and [13].

**Algorithm 5** Partition($\mathcal{T}$, Pr, $w$)

```
Input: $\mathcal{T}$, Pr, $w$
Output: $U$
1: $U \leftarrow \emptyset$
2: for i := maximum level in $\mathcal{T} \rightarrow 1$ do
3:     for node $u$ on the $i$-th level do
4:         if $\Pr(\mathcal{T}_u) > w$ then
5:             $U \leftarrow U \cup \{u\}$
6:             $\mathsf{child}_{\mathcal{T}}(\mathsf{father}_{\mathcal{T}}(u)) \leftarrow \mathsf{child}_{\mathcal{T}}(\mathsf{father}_{\mathcal{T}}(u)) - \{u\}$
7:             Adjust the corresponding weights
8:         end if
9:     end for
10: end for
11: return $U$
```

Theorem 6. *Given any instance ($\mathcal{T}$, Pr, $w$), Partition correctly solves the partition problem.*

The pseudo-code of the greedy algorithm is shown in Algorithm 6: We construct the greedy decision tree in a breadth-first manner (Lines 3-22). For each subtree $\mathcal{T}_0$, if the number of internal nodes is no more than $k$, then we simply select all of them as question nodes (Lines 5-8). Otherwise, we perform the binary search procedure (here $p_{\max} \triangleq \max\{\Pr(u) | u \in \mathcal{T}_0\}$) until we find the $k$ question nodes which best balance the partitions (Lines 9-19). We group them together as one internal node in the decision tree $\mathcal{D}$ and ask the questions for the nodes selected (Line 20). At last, we push the resulting subtrees into the queue for further operations if necessary (Line 21).

**Time Complexity:** For each loop, the number of binary searches is $O(\log n)$, each taking $O(n)$ time. There are at most $n$ loops and thus the running time is $O(n^2 \log n)$.

**Space Complexity:** $O(n)$.

**Algorithm 6** MultiGreedy($\mathcal{T}$, Pr, $k$)

```
Input: $\mathcal{T}$, Pr, $k$
Output: $\mathcal{D}$
1: $\mathcal{D} \leftarrow \emptyset$
2: Queue $S \leftarrow (\mathcal{T}, \emptyset)$
3: while $|S| > 0$ do
4:     $(\mathcal{T}_0, U) \leftarrow S.dequeue()$
5:     if $|\mathsf{Int}(\mathcal{T}_0)| \leq k$ then
6:         Ask questions for all the internal nodes in $\mathcal{T}_0$
7:         continue
8:     end if
9:     $s \leftarrow p_{\max}$, $t \leftarrow \Pr(\mathcal{T}_0)$, $mid \leftarrow \frac{s+t}{2}$
10:    $V \leftarrow$ Partition($\mathcal{T}_0$, Pr, $mid$)
11:    while $|V| \neq k$ do
12:        if $|V| > k$ then
13:            $s \leftarrow mid$
14:        else
15:            $t \leftarrow mid$
16:        end if
17:        $mid \leftarrow \frac{s+t}{2}$
18:        $V \leftarrow$ Partition($\mathcal{T}_0$, Pr, $mid$)
19:    end while
20:    Ask questions for each node in $V$
21:    Push the resulting partitions into $S$
22: end while
23: return $\mathcal{D}$
```

## 5.2 Handling Evolving Category Distribution

For practical applications, it is quite common that objects arrive in a stream fashion, and their category distribution may evolve with time. For instance, Alice wants to categorize 1,000 photos into the hierarchy in Figure 1, of which the first 400 photos she crowdsourced were taken in China, and the remaining 600 photos in Japan.

In this case, we would like our online framework to be able to capture the *evolving category distributions*. For this purpose, we extend our basic online approach by only keeping the information within a fixed size window (i.e., FPL with a window, FPLW). Consider the basic setting where we only process one object in each stage [**], and denote by $o_1, o_2, \ldots, o_n$ the sequence of objects, and by $\alpha$ the fixed window size which serves as a parameter. The pseudocode of this strategy is shown in Algorithm 7. The only difference between FPLW and FPL appears when $t > \alpha$, where FPLW deletes the categorized results outside the window while FPL keeps all the previous information.

**Algorithm 7** FPLW($T, \mathcal{O}, \lambda, \alpha$)

```
Input: $T, \mathcal{O}, \lambda, \alpha$
 1: $t \leftarrow 1$
 2: while $t \leq |\mathcal{O}|$ do
 3:     Take in $o_t$ from $\mathcal{O}$
 4:     $t \leftarrow t + 1$
 5:     for $u \in T$ do
 6:         Choose $q(u) \sim \mathsf{exp}(\lambda)$
 7:         $w(u) \leftarrow c(u) + q(u)$
 8:     end for
 9:     Normalize the weight $w$ of each node as probability
10:     Construct the decision tree $\mathcal{D}$ by Greedy (or FPTAS)
11:     Ask questions on the crowd based on $\mathcal{D}$
12:     Collect the target category $\mathsf{tar}(o_t)$
13:     $c(\mathsf{tar}(o_t)) \leftarrow c(\mathsf{tar}(o_t)) + 1$
14:     if $t > \alpha$ then
15:         $c(\mathsf{tar}(o_{t-\alpha})) \leftarrow c(\mathsf{tar}(o_{t-\alpha})) - 1$
16:     end if
17: end while
```

## 6. EXPERIMENTAL EVALUATION

In this section, we perform an extensive experimental study and the goals of our experiments are: (a) to validate the framework we proposed in the paper; (b) to compare our algorithms with other approximation algorithms. In Section 6.1, we introduce the setup of the experiment. In Section 6.2, we perform synthetic data evaluation to examine the effectiveness of our decision tree algorithm and the proposed framework. Finally in Section 6.3, we evaluate our approaches on a real crowdsourcing platform figure-eight.com.

## 6.1 Evaluation Setup

**Configuration:** All the algorithms were implemented in Python and the experiments were run in memory on an Intel i7-4870HQ CPU at 2.5GHz with 16 GB RAM.

**Data Set:** We used two real datasets.

Amazon [23] is a large product dataset with more than 15 million items. Each record in the metadata file (`https://nijianmo.github.io/amazon/index.html`) contains information such as title, price, description and category of a product sold on Amazon. The attribute "category" is a top-down path starting from the root of the hierarchy. After preprocessing, we constructed the category tree from all the paths which turned out to have 22,857 nodes.

[**]This could be trivially extend to the batched algorithm.

Table 1: Hierarchy statistics

| dataset | depth | #leaves | avg leaf depth | avg internal node degree | max degree |
|---|---|---|---|---|---|
| Amazon | 9 | 18477 | 4.60 | 5.22 | 155 |
| ImageNet | 12 | 21479 | 5.84 | 4.44 | 402 |

ImageNet[††] [6] is a large-scale hierarchical image dataset with over 12 million labeled images. ImageNet uses the hierarchical structure of WordNet(`wordnet.princeton.edu/`) and the structure file is available at `www.image-net.org/api/xml/structure_released.xml`. Each category (formally called a "synonym set" or "synset") in the hierarchy is described by multiple words/synonyms in the attribute *words* expressing a distinct concept. We extract all categories except the one with *wnid* "fall11misc" which lacks compliance with WordNet structure. This hierarchy has 27,714 nodes.

Statistics of the two hierarchies are listed in Table 1.

**Decision Tree Algorithms:** We compared the expected cost of the greedy algorithm to those of other algorithms executed on the same input hierarchy and the category distribution. In each round, $k$ questions are asked for each object. We implemented the following algorithms:

Greedy: The proposed greedy algorithm in this paper, of which the strategy is to choose $k$ questions to minimize the heaviest resulting group.

GreedyBinary: In Greedy, each question is a multiple-choice question with several options, while in IGS, all questions only have two options (Yes/No). Readers may find it somehow unfair when comparing the cost of Greedy and IGS without considering the number of options because Greedy can obtain more information from the answer of a multiple-choice question. Thus, we also evaluate the performance of the binary version of Greedy, to eliminate such doubts. The basic idea of this version is also to greedily select $k$ nodes to produce questions to ask. In the original Greedy, all edges incident on selected (internal) nodes are removed, while in binary version for each selected node (a leaf node is also legitimate here) only the edge connected to its father is removed, and thus each node corresponds to a yes-or-no question. The criterion remains the same, i.e., to make the heaviest resulting subtree as light as possible.

IGS: [29] The state-of-the-art algorithm of interactively querying on a hierarchy, of which the basic idea is to take advantage of heavy-path decomposition.

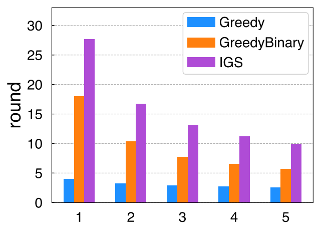
**Framework Variations:** As described in Sections 4 and 5, we proposed FPL and its extension FPLW. Moreover, various decision tree algorithms can be used in the framework.

**Framework Parameters:** The parameters of our framework evaluated in the experiments are as follows: Number of questions $k$ asked for one object at each round (Section 5). Number of arriving objects at each stage $m$ (Section 5). Memory Window Size $\alpha$: The range is $[5K, 40K]$ (Section 5).Coefficient of the exponential distribution $\lambda$: The default value is 8. The range is $[0.25, 64]$ (Section 4).

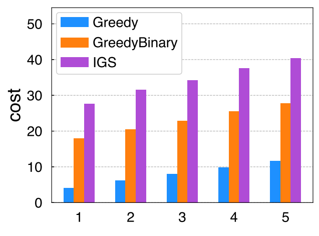
## 6.2 Synthetic Data Evaluation

In this section, we demonstrate the performance of proposed algorithms on synthetic data. When generating work-

[††]`http://image-net.org/`

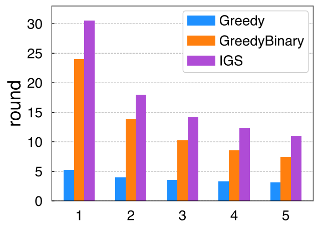


(a) round on different k

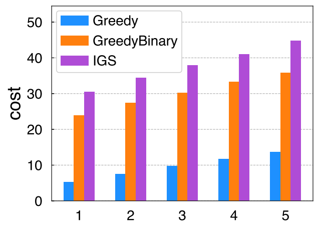


(b) cost on different k

Figure 7: Overall performance on Amazon



(a) round on different k



(b) cost on different k

Figure 8: Overall performance on ImageNet

load, let the *a-priori* probability of the hierarchy be the fraction of objects (products/images) recorded at each node.

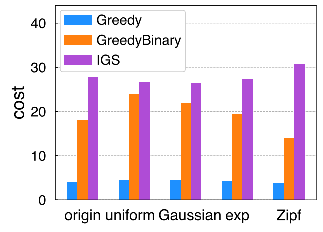
**Overall performance evaluation:** We first compare the overall performance of different decision tree algorithms. The workload consists of randomly sampled 100,000 objects from each dataset. The metrics used in this experiment are *round* and *cost*, i.e., the number of rounds (remember $k$ questions are issued in each round) and the number of all questions asked for one object until its target category is determined. It is clear that when $k = 1$ (one question per round), these two metrics are the same. By feeding the workload to the tested algorithms, the averages of *round* and *cost* over all objects demonstrate the overall performance which is of great importance. To simulate the real scenario, Greedy and GreedyBinary use FPL framework, where algorithms start from zero knowledge of the category distribution of the workload. By setting $m = 1,000$ and $\lambda = 8$, the framework gradually learns the distribution underlying the workload, and the average round and cost in the online scenario are reported.

Figures 7 and 8 show the overall performance on two datasets. In Figure 7a, Greedy takes only a small number of *rounds* to finish the workload of Amazon compared to IGS, and when $k = 1$, the ratio is below 15%. When both algorithms issue yes-or-no questions, GreedyBinary still needs fewer rounds (about 60% of IGS). It's natural that the number of rounds goes smaller with a larger $k$, as more questions are issued every time. Now we can turn to Figure 7b which shows the cost. The same result can be drawn that Greedy needs just a small cost compared to IGS, and GreedyBinary takes no more than 70% of cost of IGS. The cost exhibits an increasing trend as $k$ goes from 1 to 5, due to the limited effect of reducing rounds by batching multiple questions. It would be wise of requesters to select a good $k$ to reach the balance of latency (a larger $k$ leads to fewer rounds) and monetary cost (a smaller $k$ comes with fewer questions).

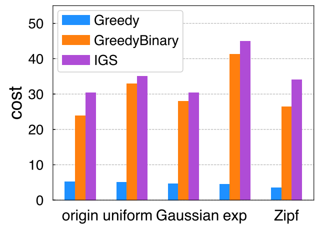
Figure 8 demonstrates the similar results. When tested on ImageNet, the round and cost ratio of Greedy to IGS are averagely 23.7% and 24.8%, and the ratios of GreedyBinary to IGS are 72.9% and 79.8%. Our proposed algorithms still have an evident advantage. Note that the ratios in ImageNet

Table 2: Statistics (*: $\times 10^5$; #: coefficient of variation)

| dataset | max frac | 20% prop | mean* | std* | $c_v$# |
|---|---|---|---|---|---|
| Amazon | 0.012 | 91.0% | 4.4 | 24 | 5.5 |
| ImageNet | 0.00031 | 63.6% | 3.6 | 4.6 | 1.3 |



(a) performance on Amazon



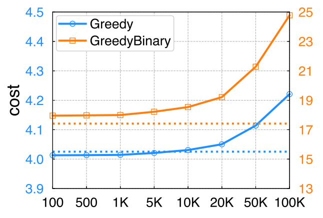
(b) performance on ImageNet

Figure 9: Evaluation on different distributions

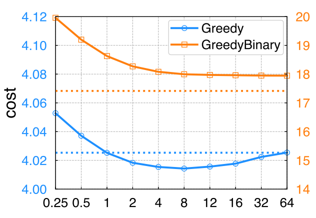
become slightly larger compared to results on Amazon, which can be partly explained by the distribution characteristics. Considering the whole workload, if we take the fraction of objects recorded at each node as its probability, a workload distribution is produced. The statistical information of two datasets is listed in Table 2. The largest fraction (the fraction of the most common category) on Amazon is 0.012, while that on ImageNet is only 0.00031. Moreover, the 20% most common categories in Amazon hold 91.0% objects while such categories in ImageNet hold only 63.7% objects. the mean, std (standard deviation) and coefficient of variation ($c_v$, the ratio of std to mean) of the workload distribution on two datasets are also displayed. In Amazon, $c_v$ is larger than that of ImageNet, which reveals the distribution are more uneven, and this is where our framework demonstrates its power to capture a dispersive distribution and make full use of it to give a satisfactory performance.

We have also conducted experiments with different a-priori distributions, to check the performance of our framework in various scenarios. The a-priori Distributions interested here are uniform, exponential, Gaussian and Zipf distributions. Given the hierarchy $\mathcal{T}$ with $n$ nodes, by traversing $\mathcal{T}$ in an in-order way (within an depth-first search, we record the first half of the subtrees of the current node, then the current node itself, and finally the second half), a sequence of $n$ nodes is obtained. For discrete distributions like uniform and Zipf (with exponent $s = 1$), we just assign corresponding probability to each node and sample 100,000 objects. For exponential (with coefficient $\lambda = 1$) and Gaussian (with mean $\mu = 0$ and variance $\sigma^2 = 1$) distribution, we select the interval which covers 99.73% probabilities ($[0, 5.91)$ and $[-3, 3)$ respectively), cut it into $n$ equi-length parts and match each part to a node in the obtained sequence in order. By sampling from the interval 100,000 times, we get 100,000 corresponding objects as our workload.

We set $k = 1$ and report the results in Figure 9, in which the distribution with the fractions as the a-priori probabilities mentioned at the beginning of this subsection is named *origin*. It is clear that under different distributions, our framework Greedy always gives the smallest cost. Moreover, GreedyBinary maintains its edge over IGS, but the cost ratio between them may vary with the distribution and/or hierarchy structure, from 45.4% (Amazon, Zipf) to 94.0% (ImageNet, uniform). A probable trend can be observed that a more skewed distribution like Zipf can help GreedyBinary acquire larger superiority, and hierarchy structure can have an influence on its performance.

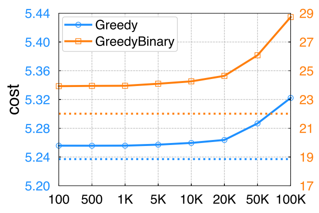


(a) varing $m$

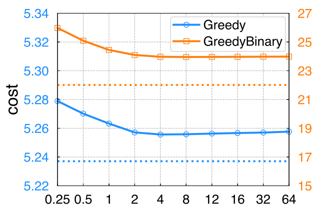


(b) varing $\lambda$

Figure 10: Evaluation on Amazon (horizontal: offline cost)



(a) varing $m$



(b) varing $\lambda$

Figure 11: Evaluation on ImageNet (horizontal: offline cost)

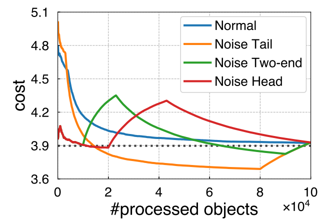
**Framework evaluation:** This experiment aims to figure out how FPL+Greedy/GreedyBinary (i.e., using FPL as the learning component and Greedy/GreedyBinary as the optimization component) captures the underlying distribution of workload and approaches the offline near-optimal performance. By using different parameters, a deeper view of algorithm performance is unveiled. All experiments in this section are set $k = 1$, and thus we only report the cost. By Theorem 1, retrieving the offline optimal cost is NP-hard. Thus, we use the offline cost (obtained by initializing the category distribution from the whole workload) incurred by our algorithm to approximate its offline optimal cost.

First by fixing $\lambda = 8$, we change the value of $m$ from 100 to 100,000, and report the result. Please refer to Figure 10a for result on Amazon with the horizontal line as the offline cost. The online cost of the framework increases as $m$ increases. It's clear to see that marginal effect of updating the estimated distribution more frequently is, indeed, quite limited when $m \leq 5,000$ (i.e, distribution are updated at least 19 times), and surprisingly the online cost of Greedy is even smaller than the offline cost in this range! So a small number of updates are sufficient to produce a satisfactory result. Thus we set $m = 1,000$ (1% of total population) by default. Even no updates are conducted ($m = 100,000$) the resulting online cost of Greedy (4.22) and GreedyBinary (24.77) maintain an obvious edge over IGS (27.66).

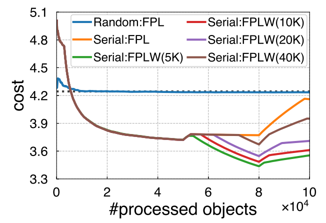
Figure 10b shows the result of varying $\lambda$ with $m = 1,000$. The online cost decreases first and then slightly goes up (Greedy) or reaches a plateau (GreedyBinary). When $\lambda = 8$, the online cost of Greedy reaches a minimum and costs are smaller than the offline cost. Figure 11 demonstrate the results on ImageNet, which have similar trends. By setting proper values of $m$ and $\lambda$, the online cost of our framework can approach to the offline cost or performs even better.

**Robustness evaluation:** We vary the workload sequence to evaluate the robustness of our framework (FPL and FPLW) with Greedy as the optimization component. We set parameters as: $k = 1$, $m = 1,000$ and $\lambda = 8$.

For the first part, we sampled 80,000 objects under the heaviest top category ("Clothing, Shoes & Jewelry" in Amazon and "artifact" in ImageNet) as *informative objects* and 20,000 objects from other categories as *noise objects* according to Pareto principle. By changing the positions of noise objects, four different sequences are generated: 1) Normal: all objects randomly distributed; 2) Noise Tail: noise objects are at the end of the workload sequence; 3) Noise Two-end: the head and tail part of the sequence are noise objects of equal length; 4) Noise Head: noise objects as the head of the workload sequence. By feeding the workload sequence to FPL, the average cost of each object is reported.

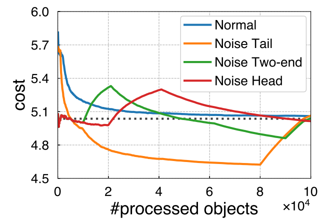


(a) varing sequence

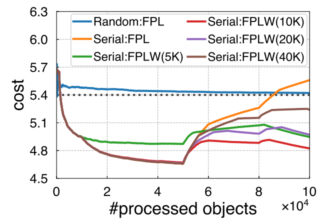


(b) varing window size

Figure 12: Robustness evaluation on Amazon, with the horizontal line as the offline cost



(a) varing sequence



(b) varing window size

Figure 13: Robustness evaluation on ImageNet, with the horizontal line as the offline cost

It can be observed from Figure 12a that for the Normal sequence, the framework gradually learns the distribution, incurs a smooth cost curve, and finally converges to the offline cost (the grey horizontal line). While for other sequences, the framework has a good beginning but when confronted with a different distribution it first incurred a lot of additional costs to absorb new information and quickly adapted itself to such change and finally approached to near the offline cost. This phenomenon provides convincing evidence that our framework is robust to variable sequences.

Next we made workload sequences of total 100,000 objects in a similar way to evaluate the performance of FPLW. Randomly sampled objects from the two heaviest categories constitute 50% and 30% of the sequence respectively with the rest from other categories. Two kinds of sequences are evaluated: 1) Random: all objects are in a random position; 2) Serial: Objects are orderly from the heaviest, the second heaviest and other categories. Performance of FPLW with different window size ($\alpha \in [5,000, 40,000]$ ) are reported.

Figure 12b shows the result of FPL and FPLW on Amazon. With Random sequence as workload, FPL incurred a good final cost near the offline cost, and for Serial workload, it made even better result. The most surprising part is that FPLW can adapt more quickly to new distributions and result in an exhilaratingly smaller cost compared to the offline one (e.g,, final cost of FPLW with window size $\alpha = 5,000$ is 3.55 vs offline cost 4.24).

Taken result on ImageNet in Figure 13 into consideration together, it's natural to see that the optimal window size may vary w.r.t dataset distribution ($\alpha = 10,000$ incurred the smallest cost in ImageNet). By carefully setting $\alpha$, our framework can achieve a promising low online cost.

Table 3: Performance on real crowdsourcing platforms

| Algorithm | Overall Accuracy | Avg Crowd Cost |
|---|---|---|
| Greedy | 92% | 22.65 |
| GreedyBinary | 84% | 86.83 |
| IGS | 80% | 151.88 |

## 6.3 Evaluation on Crowdsourcing Platforms

In this section we evaluated our proposed algorithms on a real crowdsourcing platform figure-eight.com, to demonstrate the effectiveness in real scenarios.

**Data and Workload.** We used the Amazon dataset due to its richness. To be efficient, we examined all leaf level categories (categories corresponding to leaf nodes on hierarchy), counted the number of products under each of them and randomly selected 100 products from all the products under the top 100 most popular categories as the workload.

**Algorithms.** As in Section 6.2, experiments are conducted on three algorithms: Greedy, GreedyBinary (both with FPL) and IGS. The parameters for Greedy and GreedyBinary are set: $k=1$, $m=20$ and $\lambda=8000$. All products are divided into 5 stages, when all 20 products of one stage are finished, distribution of Greedy and GreedyBinary are re-estimated and a new decision tree is constructed to proceed to next stage.

**Crowdsourcing Platform.** Figure-eight.com is a crowdsourcing platform with a great number of workers from all over the world. By exploiting the built-in functionalities of the platform the crowd quality can be controlled to a convincing degree: 1) Every worker has to pass a qualification test with accuracy no less than 85% before beginning to work on our tasks; 2) Every question is answered by at least 5 workers, and more answers are requested until a maximum of 9 answers or the minimum confidence score (0.7) is reached. Though other quality control techniques which are supplementary to our work can be used, we trade off between accuracy and cost by such platform configurations. Majority voting is adopted to aggregate answers.

**Metrics.** We used two metrics to characterize the performance: 1) *accuracy*: the rate of successfully categorized products; 2) *crowd cost*: the total number of answers collected for one product until its target category is determined. Note that each question may have 5-9 answers.

**Result.** Table 3 shows the overall performance of the tested algorithms. All algorithms have relatively high accuracy under the proper quality control, and our proposed algorithms (especially Greedy) achieve higher accuracy (92%) because fewer questions are needed and the chance of reaching incorrect categories is smaller. When we examine the average crowd cost of the successfully categorized products, our algorithms are confirmed to save a lot. Again, Greedy costs little and GreedyBinary saves more than 40% cost compared to IGS, which conform to the results on synthetic data.

Let's take a closer look at the performances of our framework at different stages in Figure 14. The average crowd cost of Greedy in all stages stay low, and there is an obvious decline trend of that of GreedyBinary due to the continuously refined distribution estimation learned from every finished stage. While IGS lacks the ability to learn from the past, it always issues a larger number of questions and performs the worst. Particularly in the last stage, the average crowd cost of IGS is about three times that of GreedyBinary.

In summary, our algorithms demonstrated the fair power

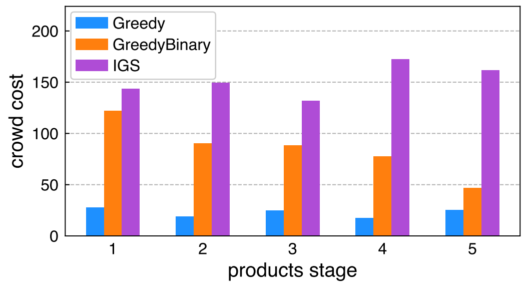


Figure 14: Cost of successfully categorized products in different stages on a real crowdsourcing platform

of estimating the category distribution from continuous learning, which brings large cost savings in a real scenario.

## 7. RELATED WORK

Comprehending images and natural languages are fairly easy for human beings, but in contrast hard for computers. Parameswaran *et al.* [25] proposed a model of utilizing human power to find target node(s) in a directed acyclic graph, which could be used for labeling large-scaled dataset in many applications. Their methods focus on shrinking the candidate set in an offline setting. Recently Tao *et al.* [29] introduced *interactive graph search* (IGS), an online problem in contrast to [25]. In IGS, the heavy path decomposition is creatively applied on the directed acyclic graph to produce an efficient question-asking strategy with appropriate interaction with the crowd. Our framework differs from theirs in the sense that we consider the category distribution of the object set and by continues learning we adaptively construct a decision tree to generate the plan of asking questions suited to the object set. To acquire labeling results of high quality with only a limited budget, existing studies [17, 9] considered to allocate redundant resources on the same object in exchange for increasing quality. Others [3, 2, 28, 16, 36, 22, 30, 15, 39, 40, 7, 38] built various workflows to utilize crowdsourcing for constructing category hierarchies.

## 8. CONCLUSION AND FUTURE WORK

We have studied the crowd-aided categorization problem. We modeled it as a decision tree construction problem and proved that constructing the optimal decision tree is NP-hard, even if the true category distribution is known. We proposed a natural greedy algorithm with an approximation ratio of 2, and also derived a fully polynomial time approximation scheme. We developed an online strategy to adaptively learn the category distribution of which the cost is near the offline optimal one. Experimental results demonstrated that our framework evidently outperformed the start-of-the-art method.

There are several interesting future directions. One promising direction is to utilize various machine learning algorithms to pre-classify the objects, which may narrow the search space and bring further savings. By modeling and exploiting the similarity between objects, we can reduce the number of tasks. Also, more task interfaces and query methods can be introduced to facilitate the cooperation between humans and machines. On the theoretical side, the approximability for constructing the optimal decision tree in the multiple-question setting remains an open problem.

**Acknowledgement.** This paper is supported by NSF of China (61925205, 61632016), Huawei, and TAL.

## 9. REFERENCES

[1] Efficient algorithms for crowd-aided categorization. `http://dbgroup.cs.tsinghua.edu.cn/ligl/crowd-cate.pdf`.

[2] J. Bragg, D. S. Weld, et al. Crowdsourcing multi-label classification for taxonomy creation. In *AAAI*, 2013.

[3] L. B. Chilton, G. Little, D. Edge, D. S. Weld, and J. A. Landay. Cascade: Crowdsourcing taxonomy creation. In *CHI*, pages 1999–2008. ACM, 2013.

[4] F. Cicalese, T. Jacobs, E. Laber, and M. Molinaro. On the complexity of searching in trees and partially ordered structures. *Theor. Comput. Sci.*, 412(50):6879–6896, Nov. 2011.

[5] A. Das Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy. Crowd-powered find algorithms. In *ICDE*, pages 964–975, March 2014.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, pages 248–255, 2009.

[7] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.

[8] Y. Gao and A. Parameswaran. Finish them!: Pricing algorithms for human computation. *PVLDB*, 7(14):1965–1976, Oct. 2014.

[9] P. G. Ipeirotis, F. Provost, V. S. Sheng, and J. Wang. Repeated labeling using multiple noisy labelers. *Data Mining and Knowledge Discovery*, 28(2):402–441, 2014.

[10] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *J. Comput. System Sci.*, 71(3):291–307, 2005.

[11] H. Kaplan, I. Lotosh, T. Milo, and S. Novgorodov. Answering planning queries with the crowd. *PVLDB*, 6(9):697–708, July 2013.

[12] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *NIPS*, pages 1953–1961, 2011.

[13] S. Kundu and J. Misra. A linear tree partitioning algorithm. *SIAM J. Comput.*, 6(1):151–154, 1977.

[14] G. Li, C. Chai, J. Fan, et al. CDB: A crowd-powered database system. *PVLDB*, 11(12):1926–1929, 2018.

[15] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 28(9):2296–2319, Sept. 2016.

[16] K. Li and G. Li. Approximate query processing: What is new and where to go? *Data Science and Engineering*, 3(4):379–397, 2018.

[17] C. Lin, D. S. Weld, et al. To re (label), or not to re (label). In *AAAI*, 2014.

[18] B. Liu. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167, 2012.

[19] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.

[20] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[21] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214, 2011.

[22] J. L. Mingda Li, Hongzhi Wang. Mining conditional functional dependency rules on big data. *Big Data Mining and Analytics*, 03(01):68, 2020.

[23] J. Ni, J. Li, and J. McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP*, pages 188–197, 2019.

[24] A. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. *PVLDB*, 7(9):685–696, May 2014.

[25] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: It's okay to ask questions. *PVLDB*, 4(5):267–278, Feb. 2011.

[26] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: Algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012.

[27] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: Declarative crowdsourcing. In *CIKM*, pages 1203–1212, 2012.

[28] Y. Sun, A. Singla, D. Fox, and A. Krause. Building hierarchies of concepts via crowdsourcing. In *IJCAI*, pages 844–851, 2015.

[29] Y. Tao, Y. Li, and G. Li. Interactive graph search. In *SIGMOD*, pages 1393–1410, 2019.

[30] S. Tian, S. Mo, L. Wang, and Z. Peng. Deep reinforcement learning-based approach to tackle topic-aware influence maximization. *Data Science and Engineering*, 5(1):1–11, 2020.

[31] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998, 2012.

[32] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071–1082, Aug. 2014.

[33] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, July 2012.

[34] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, pages 229–240, 2013.

[35] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, Apr. 2013.

[36] H. T. F. X. J. L. Yaojing Wang, Yuan Yao. A brief review of network embedding. *Big Data Mining and Analytics*, 2(1):35, 2019.

[37] C. J. Zhang, Y. Tong, and L. Chen. Where to: Crowd-aided path selection. *PVLDB*, 7(14):2005–2016, Oct. 2014.

[38] Y. Zheng, G. Li, and R. Cheng. DOCS: domain-aware crowdsourcing system. *PVLDB*, 10(4):361–372, 2016.

[39] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng. Truth inference in crowdsourcing: Is the problem solved? *PVLDB*, 10(5):541–552, 2017.

[40] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng. QASCA: A quality-aware task assignment system for crowdsourcing applications. In *SIGMOD*, pages 1031–1046, 2015.